EFFICIENT PARALLEL ALGORITHMS FOR SOME WEAK EXTERNAL VISIBILITY PROBLEMS

By

S. Veerabhadreswara Rao

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

APRIL, 1993

EFFICIENT PARALLEL ALGORITHMS FOR SOME WEAK EXTERNAL VISIBILITY PROBLEMS

A Thesis Submitted

in Partial Fulfilment of the Requirements

for the Degree of

MASTER OF TECHNOLOGY

by

S. Veerabhadreswara Rao

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

April, 1993

CSE - 1993 - M
RAO - E

CSE - 1993 - M - RAO - EFF

Certificate

Certified that the work contained in this thesis titled **EFFICIENT PARALLEL ALGORITHMS FOR SOME WEAK EXTERNAL VISIBILITY PROBLEMS** has been done by **S. Veerabhadreswara Rao** (Roll No. 9111127) under our supervision and it has not been submitted elsewhere for a degree.

[illegible]
Dr. P. Gupta
Associate Professor
Dept. of Computer Sc. & Engg.
I. I. T. Kanpur

[illegible]
Dr. A. Mukhopadhyay
Associate Professor
Dept. of Computer Sc. & Engg.
I. I. T. Kanpur

7 April, 1993

Abstract

In this thesis some weak external visibility problems have been considered. We have presented an optimal parallel algorithm for computing the shortest line segment from which a given convex polygon is weakly externally visible, and a sub-optimal parallel algorithm for computing the shortest line segment from which a given simple polygon is weakly externally visible. We also present an optimal parallel algorithm for computing the convex hull of a weakly externally visible simple polygon. The parallel computational model used by our algorithms is the CREW PRAM.

ACKNOWLEDGEMENT

I would like to express my deep sense of gratitude to my thesis supervisors Dr. A. Mukhopadhyay and Dr. P. Gupta for their guidance and encouragement throughout the thesis work. Learing how to do research from them and discussing problems with them have become some of my most joyful experiences in my M.Tech.

I would like to thank Bairagi, Dinesh, Sarkar, Riyaz, Sanjay Nath, P. L. Srinivas, Reddy and Chaitanya for their help. I would certainly like to thank all my classmates and friends who have made my stay more enjoyable. In particular, I thank Umashankar, Dev, P. T. Rao, Malle, K. V. Srinivas, T. S. Rao, Das, Srikanth, Suresh, Bandlamudi, A. Prasad, Shailaja and Saxena.

I can never thank enough my sisters, Madhavi, Uma and Sharada, and my friends, Madhavi, Rajasekhar, Srinu, and Kaja who have been the greatest source of encouragement and support.

S. Veerabhadreswara Rao.

Dedicated

To My Parents

Contents

Chapter 1

Introduction

1.1 Computational Geometry and Parallel Algorithms

Computational geometry deals with geometric problems within the framework of design and analysis of algorithms. It has applications in diverse areas such as computer-aided design, pattern recognition, computer graphics, VLSI design, image processing, vision, robotics, operation research, statistics, facilities planning and layout, etc. In the last two decades, computational geometry has attracted enormous research interest and is one of the fastest growing fields in computer science. This is because the knowledge and techniques stemming from research in this area has been successfully applied to many fields of science and engineering.

Problems in the areas of artificial intelligence, medicine, energy resource management, metereology etc. require fast computing to process a vast amount of data, often in real-time. Over the past forty years, dramatic increases in computing speed have been achieved. Most of these were due to the use of faster electronic components. Progress in this direction, however, is circumscribed by the laws of physics which put an upper bound on the speed of electronic devices. It appears that the only way around this problem is to use parallelism. The idea here is that if several operations are performed simultaneously, the time taken by any computation can be significantly reduced. Here, our computational tool is a parallel computer, that is a computer with many processing units or processors. Given a problem to be solved, it is divided into a number of subproblems. All of these subproblems are now solved simultaneously, each on a different processor. The results are then combined to produce an answer to the original problem. This is a very different model of computation from the one that has served as the basis for the sequential uniprocessor machines that have been built over the last forty years.

Visibility problems are among the most fundamental ones in computational geometry. Visibility problems arise in many application areas, such as computer graphics, robotics, computer-aided design, facility layout etc. The focus of this thesis is on developing parallel algorithms for solving some weak external visibility problems. In particular, we are interested in computing the shortest line segment from which a polygon (simple or convex) is weakly externally visible. Initial results in this direction were first obtained by Bhattacharya and Toussaint [9], who gave an optimal (sequential) algorithm for computing the shortest line segment from which a convex polygon is weakly externlly visible and then by Bhattacharya *et al* [8], who extended this result to the case of a simple polygon.

1.2 Parallel Computational Model

The commonly-accepted computational model for designing sequential algorithms is the Random Access Machine(RAM). The situation is somewhat different in parallel computing in that there is more than one accepted model of computation. However, one of the most commonly used ones is the Parallel Random Access Machine or PRAM, for short. The work described in this thesis is based on the PRAM model.

PRAMs are also known as shared-memory SIMD computers or array processors. An SIMD computer consists of some identical processors, all of which are supervised by a control unit. All the processors receive the same instruction broadcast from the control unit, but operate on different data sets from distinct data streams. For most problems, processors need to communicate among themselves during the computation in order to exchange data or intermediate results. This is done through a shared memory. Shared-memory SIMD computers can be further divided into three subclasses, depending on the policies used to resolve memory access conflicts.

1. Exclusive-Read, Exclusive-Write (**EREW**) SM SIMD computers : This model is the least powerful. In this, no two processors are allowed to read from or write into the same memory location simultaneously.

2. Concurrent-Read, Exclusive-Write (**CREW**) SM SIMD computers: In this model, multiple procesors are allowed to read from the same memory location but no two processors are allowed to write into the same memory location simultaneously.

3. Concurrent-Read, Concurrent-Write (**CRCW**) SM SIMD computers : This model is the most powerful. In this model, multiple processors are allowed to read and write from the same memory location simultaneously.

   There are various methods for dealing with concurrent writes. These policies are, (i) all processors are allowed to write, provided that the value they are attempting to store are equal, otherwise access is denied to all processors, and (ii) the smallest numbered processor is allowed to write and access is denied to all other processors.

In general. simulating a CREW PRAM or a CRCW PRAM algorithm on an EREW PRAM costs a slow-down by a logarithmic factor. Similarly, simulating a CRCW PRAM algorithm on a CREW PRAM generally costs a slow-down by a logarithmic factor. For more details on the PRAM model, see [1].

The PRAM model provides several advantages in the study of parallel algorithms. First, the PRAM seems to be suitable for studying the inherent parallelism of a problem because in this model a researcher does not need to worry about interprocessor communication and processor synchronization. secondly, the PRAM can simulate any network model algorithm within asymptotically the same time and processor bound. Due to these advantages we use PRAM model to design our parallel algorithms. All algorithms in this thesis use the CREW PRAM model.

1.3 Our Thesis

In this thesis, we present new parallel techniques for solving some weak external visibility problems, related to simple and convex polygons. It is organized as follows. In chapter 2, we give a brief survey of some previous work, related to the results of our thesis. In chapter 3, we consider the weak external visibility problem on an n-vertex convex polygon C.

We describe an optimal parallel algorithm for this problem which takes $O(log\ n)$ time, and uses $n/log\ n$ processors. In chapter 4. we consider the same problem for a simple polygon. We give a sub-optimal parallel algorithm for this problem which takes $O(log^2 n)$ time, and uses $O(n/log\ n)$ processors. Conclusion and future work is given in last chapter.

Chapter 2

Previous Work

In this chapter, we give a brief survey of some previous work. related to the results of this thesis.

Horn and Valentine [15] characterized L-sets in terms of their weak visibility properties, and such characterizations for convex and star-shaped sets have been presented by Shermer and Toussaint [19].

Avis and Toussaint [5] showed that it can be determined in $O(n)$ time whether a given a simple polygon P is edge-visible from a specified edge. Sack and Suri [18] presented a linear-time algorithm for determining all the edges of P from which it is edge-visible. Yan Ke [16] considered the problem of detecting the weak visibility of a simple polygon from an internal line segment, l. He presented an $O(nlog\, n)$ time algorithm that tests if the polygon is weakly internally visible and reports one such segment when it is. He has also given an algorithm for the query version of this problem. Given a query line segment in P, whether P is weakly visible from it can be answered in $O(log\, n)$ time with $O(nlog\, n)$ preprocessing time and $O(n)$ space. He has also shown that a shortest such line segment can be found in $O(nlog\, n)$ time.

Recently, Bhattacharya *et al* [7] considered the problem of determining the sector visibility of a polygon P. Informally, sector visibility addresses the question of external visibility along rays (or sight lines) whose angles are restricted to a sector(wedge) of specified angular width σ. Thus weak external visibility from a line corresponds to sector visibility, where $\sigma = \pi$. They present algorithms which are $\Theta(n)$ when $\sigma \leq \pi$, and $\sigma = \pi$ but require $\Omega(n \log\, n)$ time when $\pi \leq \sigma \leq 2 * \pi$.

A number of parallel algorithms have also been proposed for some other visibility problems, related to polygons. Atallah *et al* [4] presented an optimal parallel algorithm (on an EREW PRAM model) for the problem of computing the visible portion of a set of non-intersecting line segments from a point in the plane, using the 'cascading divided-and-conquer' technique. Another optimal parallel algorithm for this problem was given in [6]; this algorithm, however, assumes a CREW PRAM model and is based on the many-way divide-and-conquer technique. Chen [11] presented an optimal parallel algorithm for the case where the line segments are the edges of a simple polygon.

The *kernal* of a simple polygon P is the largest subset of P such that the entire polygon is visible from every point in that subset. Cole *et al* [12] presented an $O(log\, n)$ time optimal parallel algorithm with $O(n/log\, n)$ CREW PRAM processors for computing this kernel.

Goodrich *et al* [14] presented an $O(logn)$ time parallel algorithm with $O(n)$ CREW PRAM processors for computing the weak visibility region inside a simple polygon P from an edge of it. Goodrich *et al* [14] build a data structure for ray shooting queries inside P in $O(log\, n)$ time using $O(n)$ CREW PRAM processors. The visibility graph on the vertices of P is obtained in $O(log\, n)$ time using $O(nlog\, n + k/log\, n)$ CREW PRAM processors [14], where k is the number of edges in the visibility graph.

Chandra *et al* [10] presented a parallel algorithm for computing the complete visible polygon and the weak visible polygon inside a simple polygon P from a convex subpolygon C of P (a point p in P is completely visible from C if p is visible from every point of C, and is weakly visible form C if p is visible from at least one point of C); this algorithm takes $O(log\, n)$ time and uses $O(n)$ CREW PRAM processors.

Chen [11] presented an $O(log\, n)$ time optimal parallel CREW PRAM algorithm for computing all the edges of a given polygon P from which P is edge-visible, using $O(n/log\, n)$ processors.

Though the problem of weak internal visibility of a polygon has been well-studied, both in the sequential and parallel models of computation, hardly any result exists for the corresponding notion of weak external visibility of a polygon. Initial results in this direction were first obtained by Bhattacharya *et al* [9], who presented a linear time sequential algorithm for computing the shortest line segment from which the given convex polygon is weakly externally visibile. Subsequently, Bhattacharya *et al* [8] extended this result to the case of a simple polygon.

Chapter 3

Weak External Visibility of a Convex Polygon

3.1 Introduction

An n-vertex convex polygon C is *weakly externally visible* (*wev*, subsequently) from a line segment L, which lies outside C, if for every point x on the boundary of C there is a point y on L such that the interior of $\overline{xy}$ does not intersect the interior of C. In other words, an observer who traverses the line segment from one end to the other gets a complete view of the boundary of the polygon. Clearly, there could be many different line segments from which a polygon is weakly externally visible. For example, if the polygon is centrally symmetric there are an infinite number of such line segments.

Bhattacharya and Toussaint [9] were the first to consider the problem of weak external visibility of a convex polygon. They presented a sequential linear time algorithm for computing the shortest line segment from which such a polygon is weakly externally visible. This was extended to the case of a simple polygon by Bhattacharya *et al* [8].

In this chapter, we consider the problem of designing an optimal parallel algorithm for the same problem in the CREW PRAM model of computation.

The algorithm described in [9] seems to be inherently sequential, but a couple of pertinent observations leads us to an optimal (parallel) algorithm that runs in $O(log\, n)$ time and uses $n/log\, n$ processors.

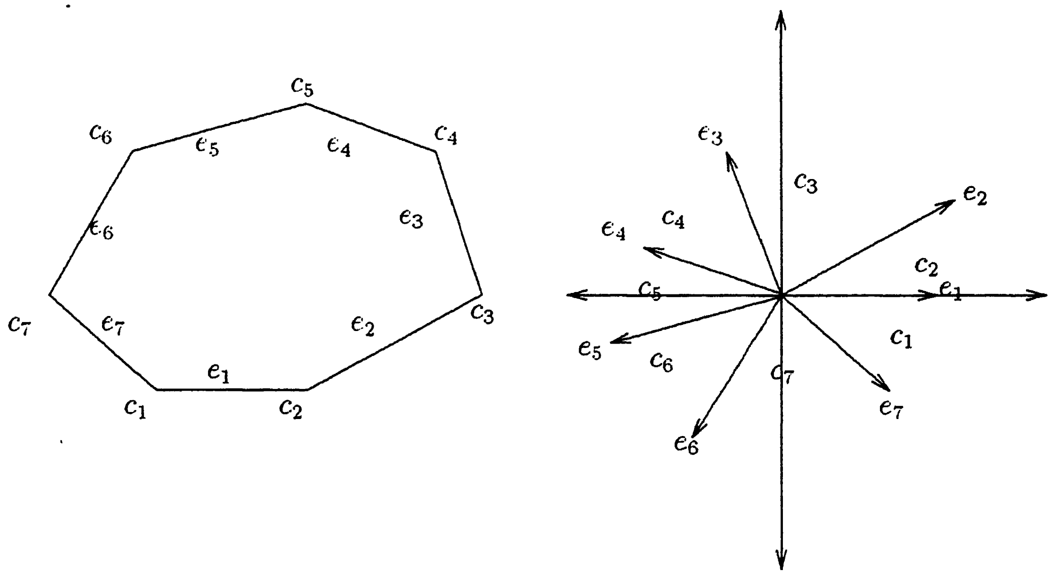

Figure 3.1: ray diagram

The rest of this chapter is organised as follows. In section 2, we introduce some notations and definitions. Section 3 contains some results on a basic geometric minimization problem and its extensions. Section 4 contains some useful observations. In section 5 we discuss a sub-optimal parallel algorithm, leading up to the optimal parallel algorithm, described in the last section.

3.2 Preliminaries

An *n-vertex convex polygon* C is specified by a sequence $(c_1, c_2, \ldots, c_n)$ of its vertices, in the order in which they are visited by a counter-clockwise walk along the boundary of C, beginning at c_1. A *chain* on the boundary of C from c_i, say, counter clockwise to $c_j, i \neq j$ is denoted by $[c_i, c_j]$. An *edge* of C, joining c_i to c_{i+1}, is denoted by $e_i = \overline{c_i c_{i+1}} (= \overline{c_{i+1} c_i})$, with the convention that $c_{n+1} = c_1$.

Translate each edge e_i so that the start vertex is at the origin of coordinates. What we get is the *ray-diagram* of the polygon, each sector representing one of its vertices. Further, the sectors occur in the same order as the vertices of the polygon. See Fig 3.1

A pair of vertices of a convex polygon that admit parallel lines of support is called an *antipodal pair*. Given a vertex c_i, the set of vertices antipodal to it forms a chain

$[c_j, c_k]$, which is called the antipodal chain of c_i. The initial and final vertices of this chain are found by drawing supporting lines to the polygon, parallel to the edges e_{i-1} and e_i respectively.

The length of an antipodal chain is the number of vertices in it and is denoted by $\|[c_j, c_k]\|$.

We denote a *ray* by $\overrightarrow{op}$, where o is the origin of the ray and p an arbitrary point on it different from o; A *cone* is denoted by $\overleftrightarrow{poq}$, where $\overrightarrow{op}$ and $\overrightarrow{oq}$ are rays with common origin o and the interior of the cone lies to right as we go from p to q via o, along the boundary of the cone. A *line segment* by $\overline{pq}$, where p and q are the end points and a *line* by $\overleftrightarrow{pq}$, where p and q are two arbitrary but distinct points on it.

We denote by $L^*(C)$ the shortest line segment from which C is *wev.*

3.3 Characterisation of the Shortest Line Segment

Our algorithm depends on a *geometric minimization* result and its variations proved in [9]. These are stated below without proof.

Lemma 1: Let p be a point lying inside a cone $W = \overleftrightarrow{qor}$ and $\overline{hk}$ a line segment through p, with end-points h, k on $\overrightarrow{oq}$ and $\overrightarrow{or}$ respectively. Then $|\overline{hk}|$ is a unimodal function of θ for $\phi < \theta < \pi$, where φ is the angle of the cone and θ is the angle that $\overline{hk}$ makes with $\overrightarrow{or}$.

The above lemma remains valid when we replace the point by a convex polygon and change other conditions in the following way.

Lemma 2: Let C be a convex polygon contained in W, such that $\overrightarrow{oq}$ and $\overrightarrow{or}$ are tangent to it. Further, let $\overline{hk}$ be tangent to the subchain of C which is concave towards o. The end points h and k lie on $\overrightarrow{oq}$ and $\overrightarrow{or}$ respectively. Then $|\overline{hk}|$ is a unimodal function of θ for $\phi < \theta < \pi$, where ϕ is the angle of the cone and θ is the angle that $\overline{hk}$ makes with $\overrightarrow{or}$

Lemma 3: $L^*(C)$ has one of its endpoints on one of the bounding rays of $\overleftrightarrow{c_{i+1}c_ic_{i-1}}$ and the other on the other bounding ray, for some value of i.

Lemma 4: $L^*(C)$ is tangent to C.

Lemma 5: $L^*(C)$ is tangent to C at c_j and in $c_{i+1}\overleftrightarrow{c_ic_{i-1}}$, then c_i and c_j are antipodal to each other.

Lemma 6: If x and y are two adjacent vertices then the last vertex in the antipodal chain of x(resp. y) is the first vertex of the antipodal chain of y(resp. x).

Proof : It can be seen easily form *ray diagram.* **Q.E.D.**

Thus for each vertex c_i there is an unique shortest line segment, which is tangent to its antipodal chain and has its end points on $\overrightarrow{c_ic_{i+1}}$ and $\overrightarrow{c_ic_{i-1}}$. So for every vertex c_i and for every vertex c_j in the antipodal chain of c_i we make use of lemma 2 to find the shortest line segment, which is tangent to the vertex c_j and in $c_{i+1}\overleftrightarrow{c_ic_{i-1}}$. The shortest of all such line segments is the required line segment.

3.4 Some Observations

In this section we present some observations, useful to the development of our algorithm.

Lemma 7: The total number of antipodal pairs in an *n-vertex convex polygon* is at most $3n/2$.

Proof : See Preparata and Shamos [17], Sec. 4.2.3.

Lemma 8: The antipodal chain of any vertex can be computed in $O(\log n)$ time using a single processor.

proof : Since an antipodal chain is made up of a contiguous set of vertices, it is enough to locate the first and last ones, which can be done in $O(\log n)$ (sequential) time by binary search. **Q.E.D.**

3.5 A Sub-optimal Algorithm

In this section, we present a basic procedure and a sub-optimal parallel algorithm.

This basic procedure *SHORT-LINE* takes an antipodal pair (c_i, c_j) as input and computes the shortest line segment, tangent to c_j, from which C is *wev*; the locus of the end-points of the line segment are determined by the edges of the polygon incident on c_i. It repeats this by interchanging the roles of c_i and c_j, and returns the shorter of the two line segments. This procedure runs in $O(1)$ time by lemma 2. We describe it formally below.

```
Procedure SHORT-LINE
Input:    An antipodal pair $(c_i, c_j)$.
Output:   The shortest line segment from which $C$ is wev with respect
          to input antipodal pair.
step 1.   Find the shortest line segment $l_1$, which is
          inside a cone $\overleftrightarrow{c_{i+1}c_ic_{i-1}}$ and tangent to $C$ at $c_j$.
Step 2.   Find the shortest line segment $l_2$, which is
          inside a cone $\overleftrightarrow{c_{j+1}c_jc_{j-1}}$ and tangent to $C$ at $c_i$.
Step 3.   Return $(\min(l_1, l_2))$.
```

It is easy to design a sub-optimal parallel algorithm, using the above procedure. Each vertex c_i is assigned a processor PE_i, which computes its antipodal chain $[c_j, c_k]$. By Lemma 8, this takes $O(log\ n)$ time. Depending on the length of this antipodal chain, we get three different cases.

Case 1: $|[c_j, c_k]| = 1$ i.e., $j = k$.

Processor PE_i invokes the *SHORT-LINE* procedure with the antipodal pair (c_i, c_j) as parameter.

Case 2: $|[c_j, c_k]| = 2$ i.e., $j + 1 = k$.

Processor PE_i invokes the *SHORT-LINE* procedure twice, once with the antipodal pair (c_i, c_j) as parameter and next with the pair (c_i, c_k).

Before discussing the third case we prove the following lemma.

Lemma 9: Let the antipodal chain $[c_j, c_k]$ of a vertex c_i have length greater than 2. Then the antipodal chain of any vertex, lying strictly between c_j and c_k, consists of c_i only.

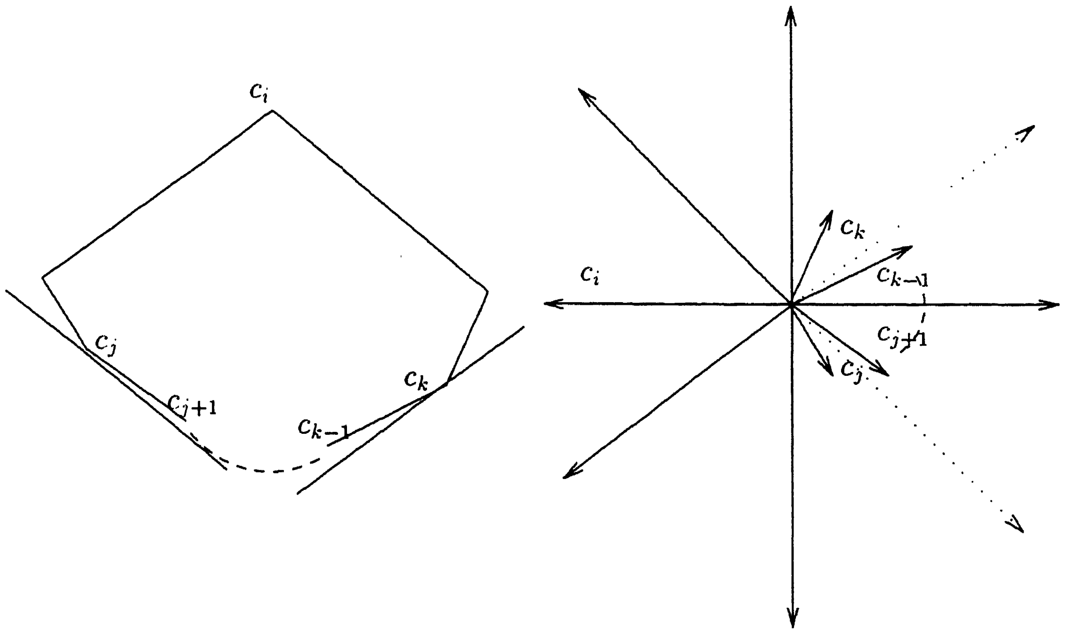

Figure 3.2:

Proof : We appeal to the ray-diagram for a neat proof of this. See Fig 3.2. To determine the antipodal chain of a vertex c_i from the ray-diagram, we extend the rays bounding its corresponding sector in this diagram in the opposite direction. This creates a sector. The vertices corresponding to the sectors that the sector so generated contain~~s~~ or intersect~~s~~ constitute the antipodal chain of c_i. Therefore, the sector that we generate to determine the antipodal chain of a vertex that lies strictly between the end-points of the chain, antipodal to c_i, is contained in the sector corresponding to c_i, which thus is its only antipodal vertex. This proves the lemma. **Q.E.D.**

Now consider the case 3.

Case 3: $|[c_j, c_k]| = m > 2$

By Lemma 9, the antipodal chain of a vertex in $[c_{j+1}, c_{k-1}]$ consists of c_i only. Therefore, the allocation of work to the processors is done as follows. Processor PE_l , where $j < l < k$, invokes the SHORT-LINE procedure on the antipodal pair (c_l, c_i), while processor PE_i takes care of the antipodal pairs (c_i, c_j) and (c_i, c_k).

Thus irrespective of the length of the antipodal chain of c_i, the associated processor invokes the *SHORT-LINE* procedure at most twice.

We give a formal description of this algorithm below.

```
Algorithm SHORTEST-$WEV$-LINE
Input:   An n-vertex convex polygon C.
Output:  The shortest line segment from which C is wev
step 1.  Assign a processor to each vertex.
Step 2.  For i := 1 to n pardo
             a[i,1]:=first vertex in the antipodal chain of $c_i$.
             a[i,2]:=last vertex in the antipodal chain of $c_i$.
Step 3.  For i := 1 to n pardo
             $l_1$:=SHORT-LINE($c_i, c_j$).
             $l_2$:=SHORT-LINE($c_i, c_k$).
             l[i]:=minimum($l_1, l_2$).
         endfor.
Step 4.  minimum(l).
```

In this algorithm step 1 takes constant time. Step 2 takes $O(log n)$ time by lemma 8. Step 3 takes constant time and step 4 takes $O(log\ n)$ time. Thus the shortest line segment can be found in $O(log\ n)$ time, using $O(n)$ processors.

3.6 Optimal Parallel Algorithm

In this section we present an optimal parallel algorithm, which takes $O(log\ n)$ time and uses $O(n/log\ n)$ processors. The essential idea of this algorithm is similar to that of the sub-optimal algorithm, presented in the previous section, with one major difference. Each processor is assigned to a group of $log\ n$ contiguous vertices instead of one.

We divide the n vertices into $n/log\ n$ groups, $G_1, G_2, \ldots, G_{n/log\ n}$, each consisting of $log\ n$ contiguous vertices. Thus the group G_i contains the vertices from $c_{(i-1)log\ n+1}$ to $c_{ilog\ n}$. Let processor PE_i be assigned to group G_i.

Each processor computes the antipodal chain of its group. That is, processor PE_i finds the first vertex in the antipodal chain of $c_{(i-1)log\, n+1}$ and the last vertex in the antipodal chain of $c_{ilog\, n}$. Let these be $a_g[i,1]$ and $a_g[i,2]$ respectively. By lemma 6 the antipodal chain of any vertex in G_i is a subchain of $[a_g[i,1], a_g[i,2]]$. The length of the antipodal chain of a group can be $O(n)$, but we will show that it is sufficient to take care of $O(log\, n)$ antipodal pairs.

Let $a_g[i,1] \in G_j$ and $a_g[i,2] \in G_k$. Depending on the number of groups between G_j and G_k, we get three cases as in the previous section.

case 1: $j = k$ i.e., $a_g[i,1]$ and $a_g[i,2]$ are in the same group. Then processor PE_i invokes the *SHORT-LINE* procedure with antipodal pairs such as (p,q), where $p \in G_i$ and $q \in G_j$. The following lemma provides a bound on the number of such pairs.

Lemma 10: Let $a_g[i,1], a_g[i,2] \in G_j$. Then the number of antipodal pairs such as (p,q), where $p \in G_i$ and $q \in G_j$ is at most $2log\, n$.

Proof : We do an angular sweep of the ray-diagram in the counter-clockwise direction, with the initial position of the sweep line coinciding with the ray corresponding to the edge $e_{(i-1)log\, n+1}$. Each ray encountered in the sweep corresponds to a new antipodal pair, and since there are $log\, n$ rays in the ray-diagram for any group the bound follows. **Q.E.D.**

The time that this takes is obviously $O(log\, n)$.

case 2: $j+1 = k$ i.e., $a_g[i,1]$ and $a_g[i,2]$ are in two consecutive groups. Processor PE_i invokes the *SHORT-LINE* procedure with antipodal pairs such as (p,q), where $p \in G_i$ and $q \in G_j \cup G_k$. The following lemma gives a bound on the number of such pairs.

Lemma 11: Let $a_g[i,1] \in G_j$ and $a_g[i,2] \in G_k$. Then the number of antipodal pairs (p,q) such that $p \in G_i$ and $q \in G_j \cup G_k$ is at most $3log\, n$.

Proof : The proof is on the lines of the proof of lemma 10. We omit the details.
Q.E.D.

case 3: There is at least one group between G_j and G_k.

It follows from lemma 9 that the antipodal chain of any group G in $G_{j+1}, G_{j+2}, \ldots, G_{k-1}$ is a subchain of G_i. Therefore the processor assigned to this group G invokes the *SHORT-LINE* procedure with its antipodal pairs, which by case 1 above is at most $2 log\, n$, so that it takes $O(log\, n)$ time. Processor PE_i invokes the SHORT-LINE procedure with antipodal pairs (p, q) such that $p \in G_i$ and $q \in G_j \cup G_k$. By lemma 10 above, it takes $O(log\, n)$ time for this.

Lemma 12: Let $a_g[i,1] \in G_j$, $a_g[i,2] \in G_k$. If there is at least one group strictly between G_j and G_k, then the number of antipodal pairs such as (p, q) where $p \in G_i$ and $q \in G_j \cup G_k$ is at most $3 log\, n$.

Proof : The proof is on the lines of the proof of lemma 10. We omit the details.
Q.E.D.

Hence irrespective of the length of the antipodal chain of a group G_i, the associated processor invokes the *SHORT-LINE* procedure $O(log\, n)$ times.

We now give a formal description of our optimal parallel algorithm below.

Algorithm `OPTIMAL-SHORTEST-`*WEV*`-LINE`

Input: *An n-vertex convex polygon C.*

Output: *The shortest line segment from which C is wev*

```
step 1.   Divide the $n$ vertices into $n/log\, n$ groups,
          each $log\, n$ vertices.  Let them call $G_1, G_2, \ldots, G_{n/log\, n}$.
Step 2.   Assign one processor to each group.
Step 3.   For i := 1 to n/log n pardo
              $a_g[i,1]$:=first vertex in the antipodal chain of $c_{(i-1)log\, n+1}$.
              $a_g[i,2]$:=last vertex in the antipodal chain of $c_{ilog\, n}$.
```

```
Step 4.   for i := 1 to n/log n pardo
              for each $(p,q)$ such that $p \in G_i$ and $q \in G_j \cup G_k$
                      call SHORT-LINE(p,q).
          endfor.
Step 5.   Find the minimum.
```

Steps 1 and 2 in the above algorithm take constant time. Step 3 takes $O(\log n)$ by lemma 8. Step 4 takes $O(\log n)$ time by lemmas 10,11.12. The minimum can be computed in $O(\log n)$ time. Hence our algorithm runs in $O(\log n)$ time.

Chapter 4

Weak External Visibility of a Simple Polygon

4.1 Introduction

In this chapter, we discuss a solution to the weak external visibility problem for a simple polygon. Bhattacharya *et al*[8] were the first to consider the problem of weak external visibility of a simple polygon. They presented a (sequential) linear time algorithm for computing the shortest line segment from which a given simple polygon is *wev*.

Their algorithm seems to be inherently sequential, but a couple of observations give us a sub-optimal parallel algorithm, which runs in $O(log^2 n)$ time using $O(n/logn)$ processors. The approach, however, is very different from that of the sequential algorithm. Henceforth by a polygon we shall always mean a simple polygon, unless otherwise stated.

The chapter is divided into seven sections. In second section, we introduce some notations, definitions and the rank tree data structure. The third section contains a characterisation of the shortest line segment from which a polygon is *wev*. In the fourth section we present a parallel algorithm for computing the convex hull of a polygon. The fifth section contains a discussion on envelope computation. In the sixth section we discuss how to compute a locally shortest line segment. In the final section we present our parallel algorithm.

4.2 Preliminaries

In this section we present some notations, definitions and a data structure called the *rank tree*, which is used for efficient implementation of some of the parallel operations, required by our algorithm.

4.2.1 Notation and Definitions

An *n-vertex simple polygon* P is specified by a sequence $(p_1, p_2, \ldots, p_n)$ of its vertices, in counter-clockwise order along the boundary. A portion of the boundary of a polygon P from p_i counter-clockwise to p_j is denoted by $[p_i, p_j]$. An *edge* of P, joining p_i to p_{i+1}, is denoted by $e_i = \overline{p_i p_{i+1}} (= \overline{p_{i+1} p_i})$. By convention $p_{n+1} = p_1$.

The *convex hull* of a polygon P is denoted by $CH(P)$. The vertices of $CH(P)$ are specified by the sequence $(c_1, c_2, \ldots, c_k)$ of its vertices in counterclockwise order along the boundary. The angle made by a ray $\overrightarrow{op}$ with the positive x-axis is denoted by $\alpha(\overrightarrow{op})$.

A *pocket* of P is the region bounded by an edge $\overline{c_i c_{i+1}}$ and the subchain $[c_i, c_{i+1}]$ of P.

4.2.2 Rank Tree Data Structure

The rank tree, data structure which supports the operations of parallel search, concatenation and split, is described in[11]; a similar data structure, called the *hull tree*, is described in[13].

A rank tree, $RT(E)$, is a binary search tree with a set of edges E stored in its leaves in some specified order (e.g. by polar angle. If it stores points, then the points are stored in order of increasing x-coordinates). Without loss of generality, assume that edges are stored in the leaves, and by increasing (or decreasing) order of polar angles. The leaves of $RT(E)$ are doubly-linked together. Each internal node v of $RT(E)$ has four feilds; the first stores the number of leaves in $RT(E_v)$, the second stores the edge e such that e has the smallest (or largest) polar angle among the edges stored in the leaves of the $RT(E_v)$, and the other two respectively store the predecessor and the successor of e in the ordered set of edges.

We denote the height of $RT(E)$ (equal to the length of the longest root-to-leaf path in $RT(E)$) by $h(RT(E))$. Then in $O(h(RT(E))$ time, a processor can search in $RT(E)$ for an edge with a given polar angle or search for the i^{th} ranked edge stored in $RT(E)$ (i.e., the i^{th} leaf of $RT(E)$ in the left-to right order) using the first or the second feild stored in the internal nodes. For more details see[11].

4.3 Characterisation of the Shortest Line Segment

Our algorithm depends on the generalisation of a certain *geometric minimization* result and some other related results proved in[8]. These are stated below without proof.

Lemma 1: Let p be a point inside a concave cone W, formed by two polygonal chains $oq = [oq_1q_2q_3 \dots q_m]$ and $or = [or_1r_2r_3 \dots r_n]$ and $\overline{hk}$ a line segment through p, with end-points h, k on oq and or respectively. Then $|\overline{hk}|$ is a unimodal function of the angle θ that $\overline{hk}$ makes with $\overrightarrow{op}$; $\theta \in [\alpha, \beta]$, where α and β are, respectively, the angles that the tangents from p to or and op make with $\overrightarrow{op}$.

The above lemma remains valid when we replace the point inside the cone by a convex polygon and change other conditions in the following way.

Lemma 2: Let C be a convex polygon lying inside a concave cone W bounded by two convex polygonal chains $oq = [oq_1q_2 \dots q_m]$ and $or = [or_1r_2 \dots r_m]$, with o being a vertex of C. If $\overline{hk}$ is a line segment lying inside the wedge, touching P, with its end points h and k lying on oq and or respectively, then $|\overline{hk}|$ is a unimodal function of θ, where θ is the angle that $\overline{hk}$ makes with some ray $\overrightarrow{op}$ lying inside W; $\theta \in [\alpha, \beta]$, where α and β are the angles that $\overline{hk}$ makes with $\overrightarrow{op}$ when it is tangential to oq and or respectively.

Lemma 3: The shortest line segment $\overline{hk}$ from which a polygon P is *wev* is tangent to its convex hull $CH(P)$.

Let $\overline{hk}$ be the shortest line segment from which a polygon P is *wev*. Assume that $\overline{hk}$ touches P at z, belonging to the antipodal chain of p. These two points split the boundary of P into two chains $[p, z]$ and $[z, p]$.

It is quite obvious that the vertices of the chain $[p, z]$ are visible from $\overline{hz}$ only. This implies that all the external cones of support of the vertices belonging to the chain $[p, z]$ intersect $\overline{hz}$. Since $\overline{hz}$ is of minimum length, h lies on the upper envelope (with respect to z) of the CCW rays of the vertices of the chain $[p, z]$. We denote this upper envelope by CCW-envelope$[p, z]$.

A similar argument shows that k lies on the upper envelope of the CW rays of the vertices of the chain $[z, p]$. This upper envelope we denote by CW-envelope$[z, p]$.

In fact, one can make a stronger statement as the following lemma shows.

Lemma 4: Let $[s_i, t_i]$ denote the antipodal chain of the vertex c_i, s and t be any two vertices of $[c_i, s_i]$ and $[t_i, c_i]$ respectively. The shortest line segment touching the convex chain $[s_i, t_i]$ with end points resting on CW-envelope$[s, c_i]$ and CCW-envelope$[c_i, t]$ respectively is the shortest line segment antipodal to c_i from which the polygon is *wev*.

The problem of computing the shortest line segment, therefore, is to determine the point of contact and the end points h and k. The point of contact is a vertex of the convex hull of P and the end points lie on a pair of CW and CCW envelopes as in the lemma above.

4.4 Convex Hull Computation

In this section we discuss a parallel algorithm for computing the convex hull of a *wev* polygon P. Let p_l and p_r be two vertices of P with the smallest and largest x-coordinate respectively. Clearly, p_l and p_r are on $CH(P)$. The line segment $\overline{p_l p_r}$ divides $CH(P)$ into two parts, an upper hull $UH(P)$ and a lower hull $LH(P)$. We will concentrate on the problem of computing $UH(P)$, as the method for computing $LH(P)$ is exactly similar.

Given two disjoint upper hulls $UH(R)$ and $UH(S)$, the common tangent T, such that both $UH(R)$ and $UH(S)$ are below, T is called the upper common tangent of

$UH(R)$ and $UH(S)$. The following observations are useful in computing the upper hull.

Lemma 5: Let p_l and p_r be two vertices of P with the smallest and the largest x-coordinate respectively. A vertex p of the polygonal chain $[p_r, p_l]$ that lies on or below $\overline{p_l p_r}$ is not a vertex of $UH(P)$.

Proof : The proof follows from a convexity argument. Q.E.D.

For any i, j and k such that $r \leq i \leq j \leq k \leq l$, assume that $UH([p_i, p_j])$ and $UH([p_j, p_k])$ have been computed and the common tangent T touches $UH([p_i, p_j])$ at p_{t_1} and $UH([p_j, p_k])$ at p_{t_2}. Since P is a *wev* polygon there exists a ray emanating from p_j that does not intersect the polygon P and separates the polygonal chains $[p_{t_1}, p_j]$ and $[p_j, p_{t_2}]$.

Lemma 6: For any i, j and k such that $r \leq i \leq j \leq k \leq l$, assume that $UH([p_i, p_j])$ and $UH([p_j, p_k])$ have been computed and given in rank trees RT_1 and RT_2 respectively. Then the upper convex hull $UH([p_i, p_k])$ can be computed in time $h = height(RT_1) + height(RT_2)$, using a single processor.

Proof : By using binary search, starting from p_j on the edges of upper hulls, we can find the upper common tangent T in time $h = height(RT_1) + height(RT_2)$, using a single processor. Assume that T touches $UH([p_i . p_j])$ at p_{t_1} and $UH([p_j, p_k])$ at p_{t_2}. Split the rank trees and merge the required rank trees; this can done in time proportional to $h = height(RT_1) + height(RT_2)$. Q.E.D.

Combining the above observations with the optimal parallel algorithm for computing the convex hull of a sorted point set[13][11], we can compute the convex hull of a *wev* polygon P in optimal time, that is, $O(d)$ time with $O(n/d)$ processors. It must be noted that we store the points corresponding to the vertices of our polygonal chain in the leaves of the rank tree in the order in which these appear along the boundary of the chain when we traverse it in, say, counter-clockwise order.

4.5 Envelope Computation

In this section we present an optimal parallel algorithm for computing envelopes. Before that, we present an algorithm for computing CW and CCW rays which is required for envelope computation.

4.5.1 Computation of Rays

In this section, we present an optimal parallel algorithm for computing the external cone of support of each vertex of the polygon P. In other words, we compute the CW and CCW rays for each vertex.

Chen[11] presented an $O(d)$ time optimal parallel algorithm for detecting the weak visibility of a polygon from an edge using $O(n/d)$ processors, where $d \geq \log n$. We make use of this algorithm.

Using the parallel prefix technique, we divide the boundary of the polygon into groups of contiguous vertices such that : (i) the length of each group is greater than or equal to $\log n$; (ii) no pocket is divided in the middle i.e., the first and last vertices of any group are convex vertices. Assume that the i^{th} group contains less than or equal to $c \log n$ vertices for some $c \leq n/\log n$. We assign c processors to the i^{th} group and compute the CW and CCW rays for each vertex in that group using Chen's algorithm.

We formally sketch the algorithm below.

```
Algorithm RAYS-COMPUTATION
Input:    An n-vertex simple polygon C.
Output:   CW and CCW rays for each vertex p.
Step 1.   Assign O(log n) contiguous convex vertices to each processor.
Step 2.   For i= 1 to k/log n pardo
              For j=(i-1)log n+1 to i log n do
                  N[j]:=simple polygon chain length from $c_j$ to $c_{j+1}$
Step 3.   PREFIX-SUM(N).
Step 4.   For i= 1 to k/log n pardo
              For j=(i-1)log n+1 to ilog n do
                  If ((trunc(N[j]/log n))<(trunc(N[j+1]/log n))
                  then M[j] := 1 else M[j] := 0.
```

```
Step 5.    PREFIX-SUM(M).
Step 6.    For i= 1 to k/log n pardo
               For j=(i-1)log n+1 to ilog n do
                   If (M[j-1] $\neq$ M[j]) then
                   Q[M[j],1] :=trunc(N[j+1]/log n)-trunc(N[j]/log n)
                   Q[M[j],2] := j.
Step 7.    For i= 1 to k/log n pardo
               For j=(i-1)log n+1 to i log n do
                   Assign Q[j,1] processors for the group
                   of vertices from $c_{Q[j-1,2]}$ to $c_{Q[j,2]}$.
                   Apply Chen's algorithm for each group.
```

In the above algorithm step 1 takes $O(1)$ time while each of the steps from 2 to 6 takes $O(log\ n)$ time, using $k/log\ n$ processors. Chen's algorithm takes $O(log\ n)$ time for each group and therefore the above algorithm takes $O(log\ n)$ time, using $O(n/log\ n)$ processors.

The rest of this chapter develops the computational machinery required for envelope computation.

We divide the boundary of the polygon P into chains of length $log(n)$ and assign a processor to each chain. Thus the i^{th} polygonal chain $[p_{(i-1)log\ n+1}, p_{ilog\ n}]$ is assigned to the processor PE_i.

Let $n_i(\geq 0)$ be the number of vertices of $CH(P)$ in the i^{th} polygonal chain which are indexed from i_1 to i_{n_i}. Processor PE_i finds the shortest line segment, which is tangent to the antipodal chain of convex vertices in its assigned chain. To do this each processor requires its own CW and CCW envelopes. We explain how to compute CCW-envelopes only, as the method for computing CW-envelopes is exactly similar. The following observations come in handy.

Let $e_i'(=\overline{p_i'p_i''})$ be an edge of a CCW-envelope which is part of the ray CCW[i]. Such a ray is split into three parts, an initial part which precedes the edge e_i' and is

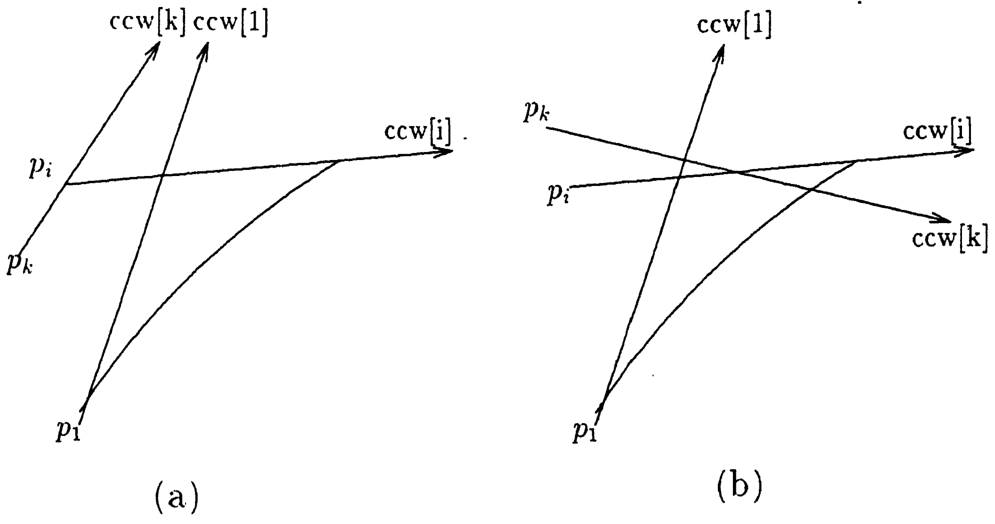

Figure 4.1:

denoted by *bef(e_i')*), the edge e_i' and a final part, which is a ray, following the edge e_i' and denoted by *aft(e_i')*).

Lemma 7: Let the CCW-ray of vertex p_{j+1}, that is CCW[$j+1$], intersect the CCW-envelope[$1,j$]. If CCW[i] is the last CCW-ray which contributed an edge to this envelope, where $1 \leq i \leq j$, then α(CCW[i]) $>$ α(CCW[$j+1$]).

Proof : Without loss of generality, we assume that the first edge in the CCW-envelope[$1,j$] is e_1'. From the definition of a CCW ray, all the vertices of polygon P lie to the left of CCW[1]. Any vertex p_k for $k > i$, is either to the left or to the right of the line(CCW[i]).

If p_k is to the right of the line(CCW[i]) then CCW[k] is bounded by the vertex p_i. Therefore α(CCW[i]) $<$ α(CCW[k]) and CCW[k] does not intersect CCW[i]. Thus CCW[k] does not intersect the CCW-envelope[$1,j$]. See Fig 4.1(a).

If p_k is to the left of the line(CCW[i]) then CCW[k] intersects CCW[i] only if α(CCW[i]) $>$ α(CCW[k]). See Fig 4.1(b).

Therefore for $j > i$, CCW[$j+1$] intersects the CCW-envelope[$1,j$] if α(CCW[i]) $>$ α(CCW[$j+1$]). **Q.E.D.**

From the above lemma we can conclude that the set of CCW-rays of a polygonal chain which contribute an edge to the CCW-envelope of this chain are angularly ordered. A CCW-ray of a vertex in a polygonal chain is called effective if at any time during the incremental construction of the CCW-envelope of this chain the ray contributed an edge to it. A subset of the effective rays is a part of the final envelope. We can optimally compute effective rays by using the parallel prefix method.

Lemma 8: Two CCW envelopes can intersect at most once.

Proof : See[8].

Lemma 9: Let CCW-envelope$[1,i]$ be kept in a rank tree RT. If CCW$[j]$, for some $j > i$, is the next ray which intersects the CCW-envelope$[1,i]$, then we can compute the CCW-envelope$[1,j]$ in sequential-time $O(h)$, where $h = height(RT)$.

Proof : We use binary search to find the point of intersection between CCW$[j]$ and CCW-envelope$[1,i]$. Consider a ray CCW$[k]$ such that $k \leq i$, such that e_k' is a edge of the CCW-envelope$[1,i]$. Let q be the point of intersection between CCW$[k]$ and CCW$[j]$. Depending on the location of q, we can eliminate a part of the CCW-envelope$[1,i]$. If q is on e_k' then we have found the intersection between CCW$[j]$ and CCW-envelope$[1,i]$ (see Fig 4.2(a)). If q is on $bef(e_k')$ then the intersection between CCW$[j]$ and CCW-envelope$[1,i]$ occurs on the part of the envelope before the edge e_k'. That is, we can eliminate the part of the envelope after the edge e_k' (see Fig 4.2(b)). Similarly, if q is on $aft(e_k')$ then the intersection between CCW$[j]$ and CCW-envelope$[1,i]$ occurs on the part of the envelope after the edge e_k'. That is, we can eliminate the part of the envelope before the edge e_k' (see Fig 4.2(c)). After finding the intersection, split the rank tree and merge with the ray CCW$[j]$. The split operation can be done in time $O(h)$[11]. Therefore we can compute the CCW-envelope$[1,j]$ in $O(h)$ time with a single processor. **Q.E.D.**

Lemma 10: Let (R_1, R_2) be two angularly ordered collection of CCW-rays. Given CCW-envelope(R_1) and CCW-envelope(R_2) in rank trees RT_1 and RT_2 respectively, we can merge the two envelopes in sequential-time $O(h)$, where $h = height(RT_1) + height(RT_2)$.

Proof : By Lemma 8, two CCW-envelopes can intersect at most once. So we can use double binary search, that is simultaneously use binary search on both the envelopes. We can compute the point of intersection in $O(h)$ time. Therefore we can merge the two envelopes in $O(h)$ time with a single processor. **Q.E.D.**

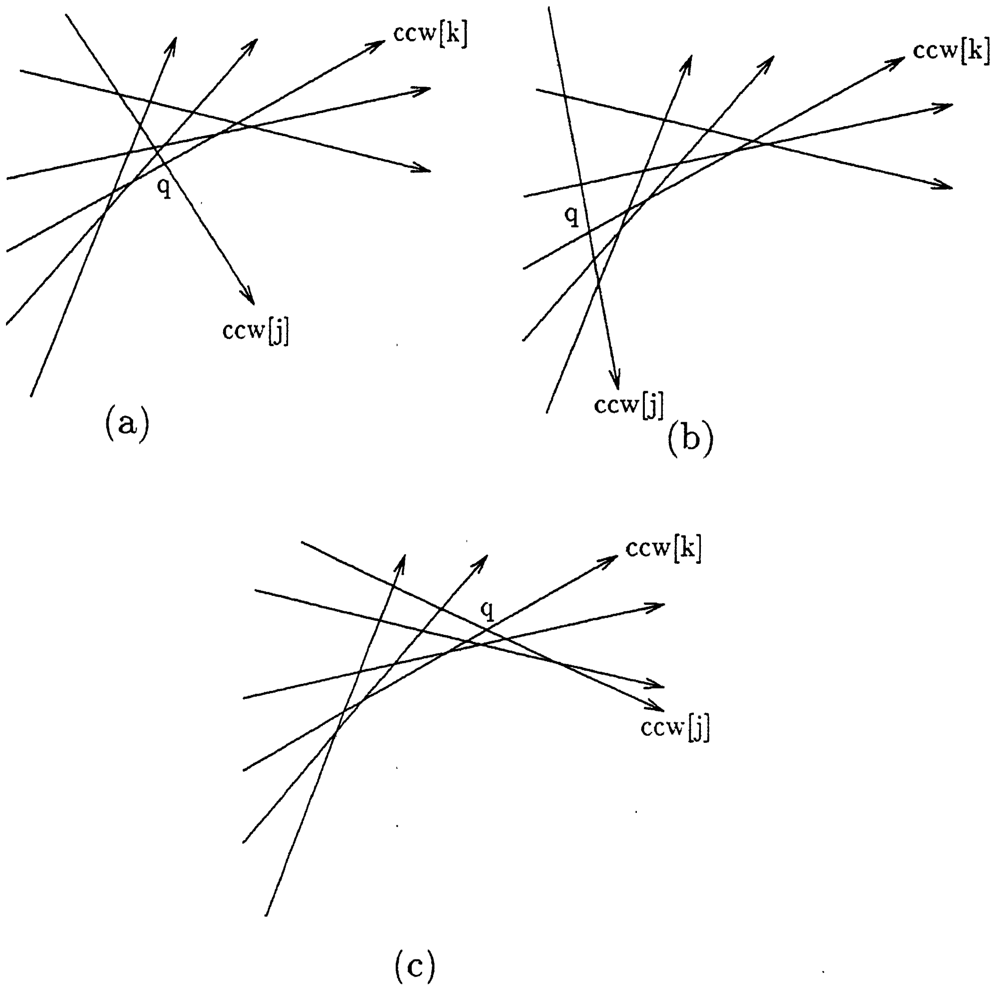

Figure 4.2:

Lemma 11: If RT_1 and RT_2 are rank trees corresponding to the envelopes CCW-envelope$[1,i]$ and CCW-envelope$[i+1,j]$, $1 \leq i \leq j \leq n$, then we can merge the envelopes in time $O(h)$, where $h = height(RT_1) + height(RT_2)$.

Proof : The proof is exactly along the lines of the proof of lemma 10. We omit duplicating the details. **Q.E.D.**

In the following lemmas we assume that we have more than one processor available to us.

Lemma 12: Let us assume that the CCW-envelope$[i_{j-1}+1, i_j]$ has been computed and given in RT_j for $j = 1, \ldots, m$ such that $i_{j-1} < i_j$ and $i_m \leq n$ with the convention that $i_0 = 0$, then for any $j = 1, \ldots, m$ we can construct a rank tree for CCW-envelope$[i_{j-1}, i_j] \cap$ CCW-envelope$[1, i_m]$ in $O(h + logm)$ time using m processors,

$$where\, h = \max_{1 \leq j \leq m} \{height(RT_j)\}$$

Proof : Our method for constructing the rank tree follows the treatment in[13]. Assign a processor to each pair (j,k) for $k = 1, \ldots, j-1, j+1, \ldots, m$. Using lemma 11, we find the intersection between RT_j and RT_k. Let $E_{j,k}$ be the edge on RT_j containing the intersection point. It takes $O(h + logm)$ time ($O(logm)$ time is required to compute the value of h). Let E_j be the edge with smallest slope in $\{E_{j,1}, E_{j,2}, \ldots, E_{j,j-1}\}$ and F_j be the edge with largest slope in $\{E_{j,j+1}, E_{j,j+2}, \ldots, E_{j,m}\}$. E_j and F_j can be found in $O(log\, m)$ time, using m processors. If the slope of E_j is greater then or equal to the slope of F_j then the part of the envelope from E_j to F_j is the required envelope, otherwise it is a null envelope. **Q.E.D.**

Lemma 13: Let us assume that the CCW-envelope$[i_{j-1}+1, i_j]$ has been computed and given in rank tree RT_j for $j = 1, \ldots, m$ such that $i_{j-1} < i_j$ and $i_m \leq n$ with the convention that $i_0 = 0$. Then the CCW-envelope$[1, i_m]$ can be computed using m^2 processors in $O(h + log\, m)$ time,

$$where\, h = \max_{1 \leq j \leq m} \{height(RT_j)\}$$

Proof : Assign m processors to each RT_j and construct a rank tree RT_j' for $RT_j \cap$CCW-envelope$[1, i_m]$ in $O(h + log\, m)$ time using lemma 12. Construct a rank tree RT, by building a complete binary tree on top of RT_j. It takes $O(log\, m)$ time. Therefore we can compute the CCW-envelope$[1, i_m]$ in $O(h + log\, m)$ time with m^2 processors. **Q.E.D.**

Lemma 14: Let us assume that the CCW-envelope$[i_{j-1}+1, i_j]$ has been computed and given in RT_j for $j = 1, \ldots, m$ such that $i_{j-1} < i_j$ and $i_m \leq n$ with the convention that $i_0 = 0$, then for $j = 1, \ldots, m$ we can compute the CCW-envelope$[i_{j-1} + 1, i_m]$ using $O(m^3)$ processors in $O(h + log\, m)$ time,

$$where\; h = \max_{1 \leq j \leq m} \{height(RT_j)\}$$

Proof: Assign $O(m^2)$ processors to each $j = 1, \ldots, m$ and construct the rank tree RT_j for the CCW-envelope$[i_{j-1}, m]$ using lemma 13. **Q.E.D.**

Our method of constructing CCW-envelopes is the following. We call procedure **CCW-ENVELOPE**, defined below, passing to it the set of CCW-rayes **R** and the integer $log\, n$, where $n = |R|$. Our algorithm is based on lemma 15 and the divide-and-conquer method of[11].

```
Algorithm CCW-ENVELOPES
Input:    A set $R$ of $n$ CCW rays and an integer $d$.
Output:   $n/log\, n$ CCW-envelopes.
Step 1.   If $|R| \leq d$ then use one processor to compute the
          CCW-envelope.
Step 2.   If $d < |R| \leq d^6$ then divide $R$ into two subsets
          $R_1$ and $R_2$ of equal size, and recursively solve
          the two subproblems in parallel. Then perform the merge
          operation on the output of the recursive call.
Step 3.   If $m > d^6$ then partition $R$ into $g = (|R|/d)^{1/3}$
          subsets $R_1, \ldots, R_g$. Then recursively solve
          the $g$ subproblems in parallel. Finally perform the
          computation using lemma 15, by using the output from
          the $g$ recursive calls.
```

Lemma 15: The algorithm CCW-ENVELOPES takes $O(d)$ time using $O(n/d)$ processors. It requiries $O(n^2)$ space.

Proof : See section 5.3.1 in [11].

4.6 Local Shortest Line Segment

Let (c_i, c_j) be an antipodal pair. Let E_h denote its CCW-envelope and E_k its CW-envelope.

To begin the search, an initial placement of the line segment $\overline{hk}$ is determined as followes: If $\alpha(\overline{c_{j-1}, c_j} > 180 - \alpha(\overline{c_{i-1}, c_i})$ then it can be determined by drawing a line through c_j parallel to $\overline{c_{i-1}, c_i}$ and compute its intersection with E_h and E_k by using binary search method. Otherwise it is determined by the $\overline{c_{j-1}, c_j}$. Similarly we can find the final placement of the line segment $\overline{hk}$.

The sequential search from an initial placement to the final placement is bounded by $O(n)$. That is, there may be at most $O(n)$ edges from an initial placement to the final placement on envelopes. But the total search time over all antipodal pairs is bounded by $O(n)$. If there are more than $log\ n$ edges in the search range on the E_h then divide the envelope E_h into groups of $log\, n$ and assign a processor to each group. Find the corresponding positions in E_k. We do the same for E_k. Each processor sequentially searches its respective range in E_h and E_k.

```
Procedure LOCAL-SHORTEST-LINE
Input:    An antipodal pair $(c_i, c_j)$ and envelopes $E_h$ and $E_k$.
Output:   The shortest line segment from which P is wev with respect
          to input antipodal pair.
step 1.   Find the initial position and final position of $\overline{hk}$.
Step 2.   If number of edges on $E_h$ between initial and final
          position is greater then $log\ n$ then
                    Divide the search range such that each
                    subrange contains $log\ n$ edges, assign
                    a processor to each subrange and find the
```

```
                 corresponding position on $E_k$.
Step 3.   If number of edges on $E_k$ between initial and final
          position is greater then $log\, n$ then
                 Divide the search range such that each
                 subrange contains $log\, n$ edges, assign
                 a processor to each subrange and find the
                 corresponding position on $E_h$.
Step 4.   Find the shortest line segment in each subrange
          sequentially.
Step 5.   Return the shortest line segment over all subranges.
```

Lemma 16: Procedure LOCAL-SHORTEST-LINE takes $O(log\, n)$ time.

4.7 Parallel Algorithm

In this section we sketch the parallel algorithm. As in the case of the sequential algorithm, we divide our problem into two smaller subproblems. One subproblem is to find the shortest line segment for antipodal vertices of the vertices c_j, where $j = 1, 2, \ldots, m-1$ such that c_1 is th antipodal vertex of c_m. The main idea is to divide a polygonal chain into subsets of $O(log\, n)$ and assign a processor to each group and find the shortest line segment of *wev* with respective to that group.

```
Algorithm SHORTEST-LINE-SEGMENT
Input:    An n-vertex simple polygon P.
Output:   The shortest line segment from which p is wev with
step 1.   Find the convex hull of given simple polygon $CH(P)$.
Step 2.   Compute the CW and CCW rays for each vertex.
Step 3.   Compute the required $n/log\, n$ CW and CCW-envelopes.
Step 4.   Divide the polygon chain into $n/log\, n$ groups
          $G_1, G_2, \ldots, G_{n/log\, n}$ and assign a processor to each group.
```

```
                    corresponding position on $E_k$.
Step 3.   If number of edges on $E_k$ between initial and final
          position is greater then $log\, n$ then
                    Divide the search range such that each
                    subrange contains $log\, n$ edges, assign
                    a processor to each subrange and find the
                    corresponding position on $E_h$.
Step 4.   Find the shortest line segment in each subrange
          sequentially.
Step 5.   Return the shortest line segment over all subranges.
```

Lemma 16: Procedure LOCAL-SHORTEST-LINE takes $O(log\, n)$ time.

4.7 Parallel Algorithm

In this section we sketch the parallel algorithm. As in the case of the sequential algorithm, we divide our problem into two smaller subproblems. One subproblem is to find the shortest line segment for antipodal vertices of the vertices c_j, where $j = 1, 2, \ldots, m-1$ such that c_1 is th antipodal vertex of c_m. The main idea is to divide a polygonal chain into subsets of $O(log\, n)$ and assign a processor to each group and find the shortest line segment of *wev* with respective to that group.

```
Algorithm SHORTEST-LINE-SEGMENT
Input:    An n-vertex simple polygon P.
Output:   The shortest line segment from which p is wev with
step 1.   Find the convex hull of given simple polygon $CH(P)$.
Step 2.   Compute the CW and CCW rays for each vertex.
Step 3.   Compute the required $n/log\, n$ CW and CCW-envelopes.
Step 4.   Divide the polygon chain into $n/log\, n$ groups
          $G_1, G_2, \ldots, G_{n/log\, n}$ and assign a processor to each group.
```

```
Step 5.   For i:=1 to n/log n pardo
          5.1. Find the antipodal chain of each group.
               Let $A_i$ be the antipodal chain of group $G_i$
               (i.e., first antipodal vertex of first
               convex vertex and last antipodal vertex of the
               last convex vertex in that group.)
          5.2. If the antipodal chain length of a group is less than
               $log\, n$ then find the shortest line segment sequentially.
          5.3. If the antipodal chain length of a group is greater than
               $log\, n$ then
               5.3.1. Divide the polygon chain $A_i$ into groups
                      $A_{i_1}, A_{i_2}, \ldots$ each $log\, n$ length and assign a
                      processor to each group.
               5.3.2. Find the antipodal chain of each polygon chain
                      $A_{i_j}$.  Let them call $G_{i_1}, G_{i_2} \ldots$
               5.3.3. Compute the CW and CCW envelopes for each group $G_{i_j}$
               5.3.4. compute the shortest line segment for each group.
Step 6.   Find the shortest line segment.
```

Each processor is assigned to two polygonal chains G_{i_j} and A_{i_j}, which are antipodal to each other. The length of these chains are at most $log\ n$. Each processor computes the shortest line segment, which is tangent to the convex polygon $CH(P)$ at some vertex in the polygon chain A_{i_j}. So the maximum number of processors required is $O(n/log\ n)$

In the *CCW-ENVELOPES* algorithm we divided the polygonal chain into groups of $log\ n$. In each group the first vertex need not be a convex vertex. We need to adjust these envelopes so that the first vertex is convex. We make this adjustment, using the merge and unmerge operations. We need to perform merge operation on the CW-envelope and unmerge operation on CCW-envelope. By Lemma 11 a merging operation can be done in $O(log\ n)$ time using a single processor. For an unmerging operation, let us assume that processor PE is assigned to the polygonal chain $G = [i, k]$ and it is required to unmerge the CCW-envelope$[i, j]$, where $i \leq j \leq k$, from its CCW-envelope$[i, n]$. Processor PE sequentially computes the CCW-envelope$[j, k]$ and merges it with the CCW-envelope of the next processor. The length of the polygon chain G is less than $log\, n$. So the CCW-envelope$[j, k]$ can be constructed in $O(log n)$ time. Therefore, unmerging can be done in $O(log n)$ using a single processor.

In the above algorithm each of the steps from 1 to 3 takes $O(log\, n)$ time using $O(n/log\, n)$ processors. We can compute the antipodal chain of any contiguous group in $O(log\, n)$ time. So steps 5.1 and 5.3.2 can be performed in $O(log\, n)$ time. We can compute the envelopes for the new groups in $O(log\, n)$ time using merge operation. So step 5.3.3 can be computed in $O(log\, n)$ time.

Each processor should find the shortest line segment with respective to the assigned groups. Each processor takes one antipodal pair at a time from its groups and calls the *LOCAL-SHORTEST-LINE* procedure. In the *LOCAL-SHORTEST-LINE* procedure we have divided the search range and assigned the processors to each range. The total search range is at most $O(n)$. Therfore the maximum number of processors required is $O(n/log\, n)$. So we can perform steps 5.2 and 5.3.4 in $O(log\, n)$ time.

We need to perform merge and unmerge operation for each convex vertex in its assigned groups. Each group has a maximum $log\, n$ convex vertices. So it takes $O(log^2 n)$ time.

Lemma 17: Algorithm SHORTEST-LINE SEGMENT computes the shortest line segment from which a given simple polygon is *wev* in $O(log^2 n)$ time using $O(n/log\, n)$ processors.

Chapter 5

Conclusion

In this thesis, we have presented parallel algorithms for solving some weak external visibility problems, related to simple and convex polygons on CREW PRAM. The basis of these algorithms are very different from those used for designing the corresponding sequential algorithms.

In chapter 3 we have designed an optimal parallel algorithm for computing shortest line segment from which given convex polygon C is weakly externally visible. The algorithm is based on some properties of antipodal pairs of points. These are: (i) If the antipodal chain length of a vertex is greater then two, then all vertices in that chain, except for the first and the last, have an antipodal chain of length exactly one. (ii) The number of antipodal pairs is $O(n)$. Using these observations, we have designed an $O(log\, n)$ time parallel algorithm that requires $O(n/log\, n)$ CREW PRAM processors.

We have presented an optimal parallel algorithm for computing the external cone of support of a vertex p of a given simple polygon P. The algorithm returns this cone if P is weakly externally visible else retuns that P is not weakly externally visible. We use a techinique presented in [11]. Our algorithm takes $O(log\ n)$ time, using $O(n/log\ n)$ CREW PRAM processors.

We have presented an optimal parallel algorithm for computing the convex hull of given weakly externally visible simple polygon P. The observation involved in our algorithm is that for every vertex p on the boundary of P, there exists a ray which emanates from p and does not intersect the boundary of P. Using this observation, we have reduced our problem to the problem of computing the convex hull of a sorted set of points [11] [13].

We have also presented a sub-optimal parallel algorithm for computing the shortest line segment from which given simple polygon P is weakly externally visible. We use the rank tree data structure of [11] and a divide-and-conquer technique. Our algorithm takes $O(log^2 n)$ time, using $O(n/log\ n)$ CREW PRAM processors.

This research presented in this thesis can be extended in the following directions.

One direction is to look into the design of an optimal parallel algorithm for the problem, discussed in the previous chapter. This essentially boils down to the problem of merging and unmerging the envelopes in linear time.

Another direction that can be explored is to see whether the approach presented here can be used to design an optimal parallel algorithm to compute the shortest internal line segment from which a polygon is weakly internally visible.

A third direction is to study these problems in higher dimensions. Designing fast sequential algorithms for these problems in higher dimensions are still open.

Bibliography

[1] S.G. Akl. *The design and analysis of parallel algorithms.* Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[2] S.G. Akl. *Parallel computational grometry.* Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[3] M.J. Atallah, D.Z. Chen and H. Wagner. *Optimal parallel algorithm for the visibility of a simple polygon from a point.* Journal of ACM, pages 516-553, July 1991.

[4] M.J.Atallah, R. Cole and M.T. Goodrich. *Casecading divide-and-conquer: A technique for desiging parallel algorithm.* SIAM Journal on computing, 18(3) pages 499-532, 1989.

[5] D. Avis and G.T. Toussaint. *An optimal algorithm for determining the visibility of a polygon from an edge.* IEEE Transactions on Computers, C-30 pages 910-914, 1981.

[6] P. Bertolazzi, S. Salza, and C. Guerra, *A parallel algorithm for the visibility problem from a point.* Journal of parallel and distributed computing, 9, pages 11-14, 1990.

[7] B. K. Bhattacharya, D. Kirkpatrick and *G.T. Toussaint. Determining sector visibility of a polygon.* In Proc. of the Fifth Annual ACM Symp. on Computational Geometry, pages 247-254, 1989.

[8] B.K. Bhattacharya, A. Mukhopadhyay and G.T. Toussaint. *A linear time algorithm for computing the shortest line segment from which a polygon is weakly*

exter nally visible. In Proc. of the Workshop on Algorithms and Data Structures, Ottawa, Canada, pages 412-424,1991.

[9] B.K. Bhattacharya and G.T.Toussaint. *Computing shortest transversals.* Technical Report SOCS 90.6, McGill University, April 1990.

[10] V. Chandru, S.K. Ghosh, V.T. Rajan and S. Saluja. *Nc-algorithms for minimum link pasth and related problems.* Tech. Report, cs-90/3, computer science group. Tata Institute of Fundamental research. Bombay, India, 1991.

[11] Z.D. Chen. *Parallel techniques for paths, visibility and related problems.* Ph.D. Thesis, Purdue university Aug 1992.

[12] R. Cole and M.T. Goodrich. *Optimal parallel algorithm for polygon and point-set problems.* In Proc. of the 4th annual ACM symposim on Computaional Gometry, pages 211-220, 1988.

[13] M.T. Goodrich. *Finding the convex hull of a sorted point set in parallel.* In Information Processing Letter, pages ,Dec 1987.

[14] M.T. Goodrich, S.B. Shauck and S. Guha. *Parallel methods for visibility and shortest path problem in simple polygons.* In Proc. of the 6th Annual ACM Symposium on computational Geometry, pages 73-82, 1990.

[15] A. Horn and F.A. Valentine. *Some properties of L-sets in the plane.* Duke Mathematics Journal,vol.16, pages 131-140, 1949.

[16] Y. Ke. *Detecting the weak visibility of a simple polygon and related problems.* John Hopkins University, manuscript, 1988.

[17] F. P. Preparata and M. I. Shamos. *Comutational Geometry: An Introduction.* Springer-Verlag, New York, 1985.

[18] J. R. Sack and S. Suri. *An optimal algorithm for detecting weak visibilty of a polygon.* IEEE Transactions on Computers, C-39 pages 1213-1219, 1990.

[19] T. Shermer and G.T. Toussaint. *Characterization of convex and star-shaped polygons.* Snapshots of computational and dicrete Geometry, Tech.Rept SOCS-88.11, School of Computer Science, McGill University, June 1988.